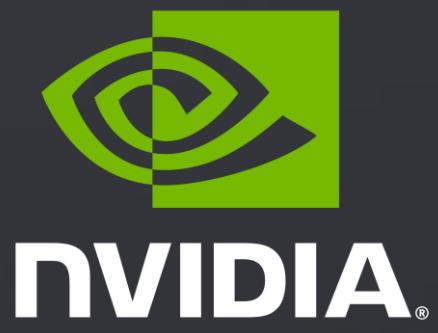


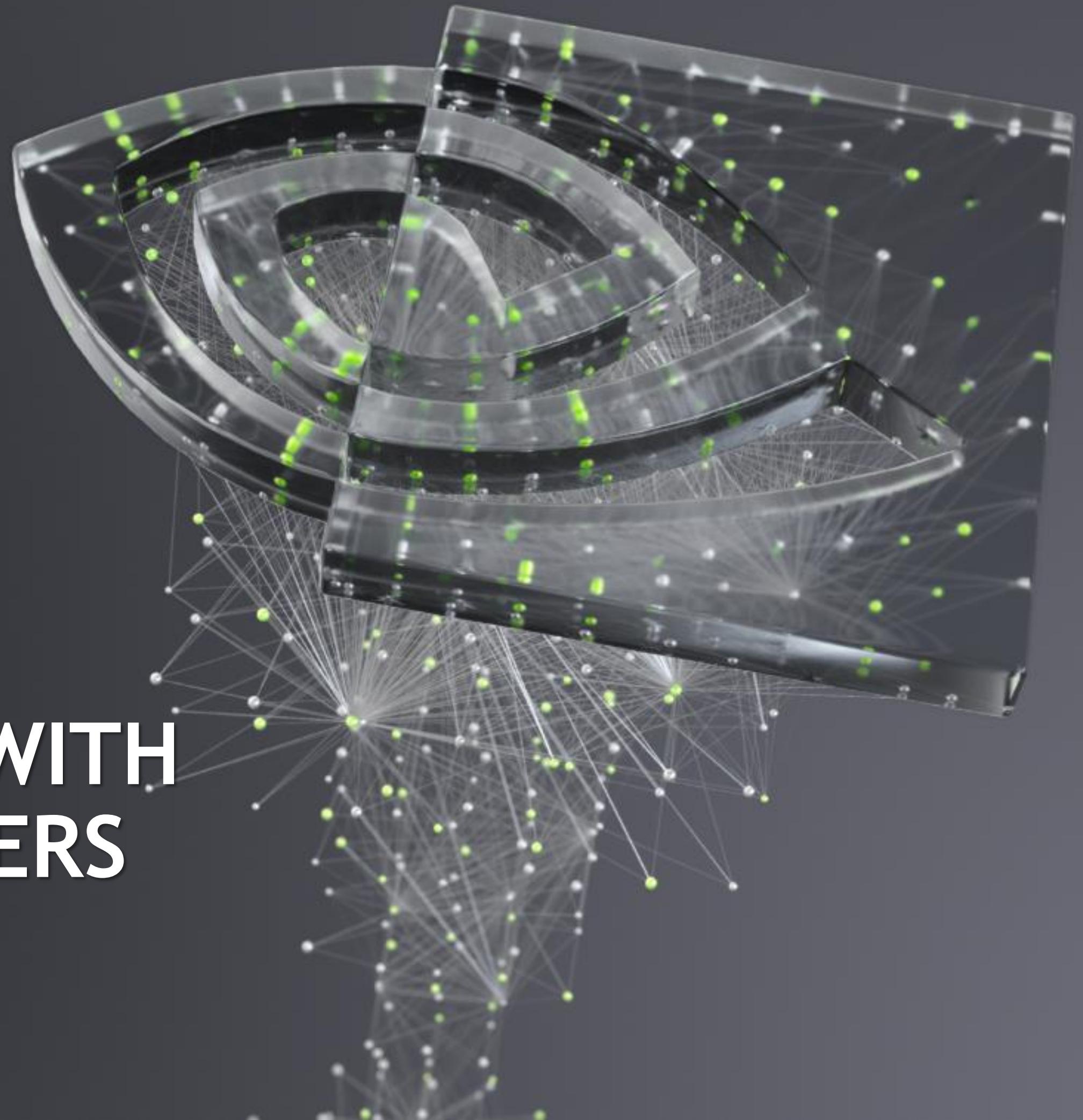
Advanced Modeling & Simulation (AMS) Seminar Series

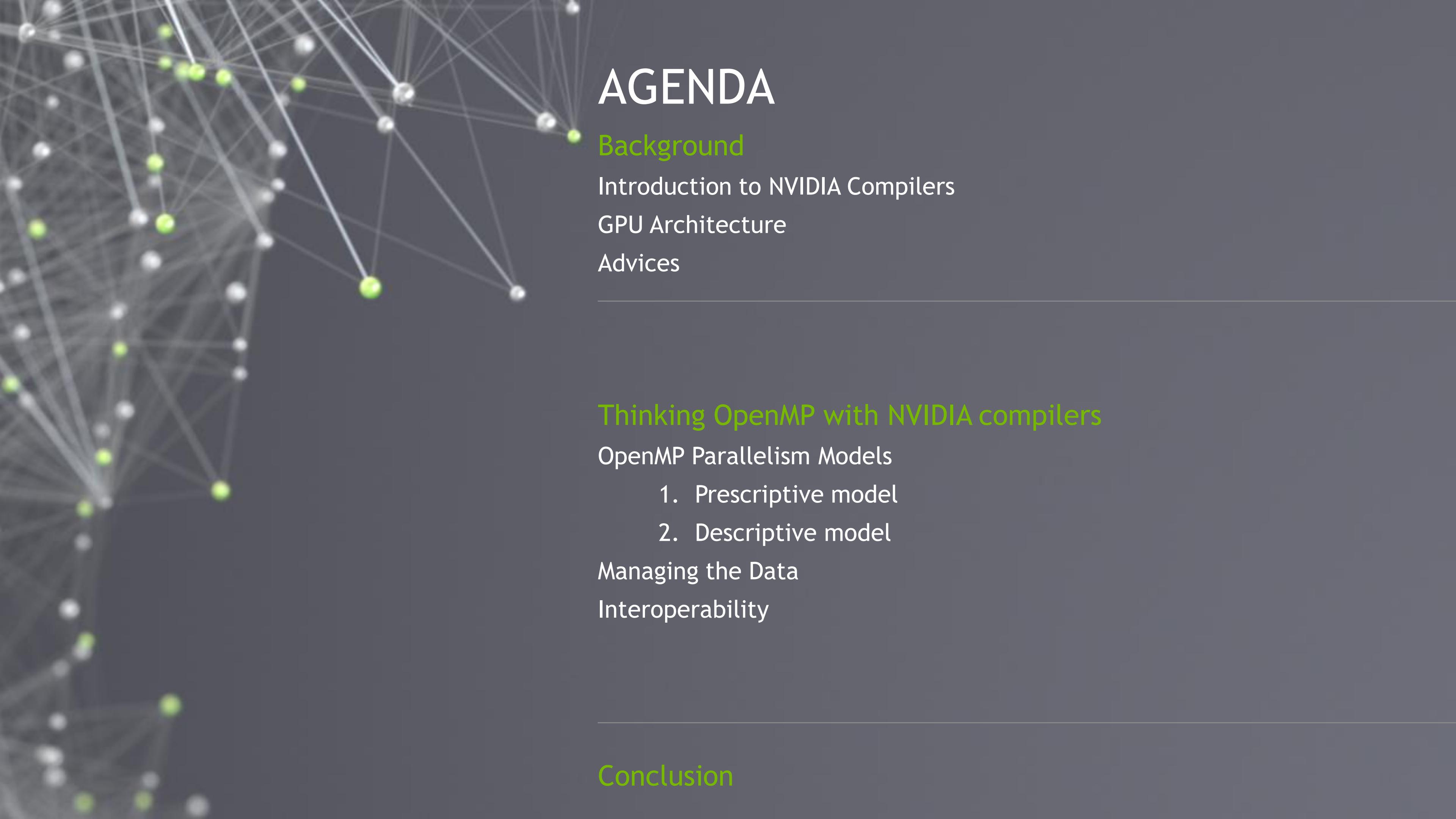
NASA Ames Research Center, May 4<sup>th</sup>, 2021



# THINKING OPENMP WITH NVIDIA HPC COMPILERS

Dr. Güray Özen, Senior Compiler Engineer





# AGENDA

## Background

Introduction to NVIDIA Compilers

GPU Architecture

Advices

---

## Thinking OpenMP with NVIDIA compilers

OpenMP Parallelism Models

1. Prescriptive model
2. Descriptive model

Managing the Data

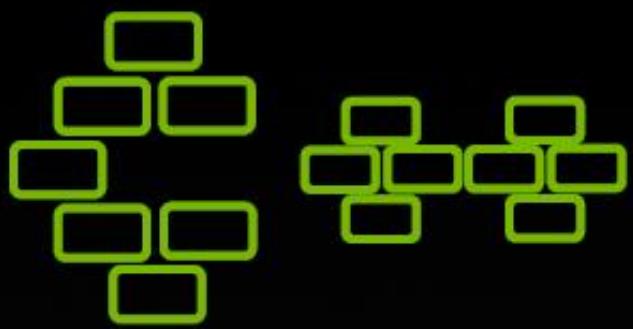
Interoperability

---

## Conclusion

# NVIDIA HPC COMPILERS

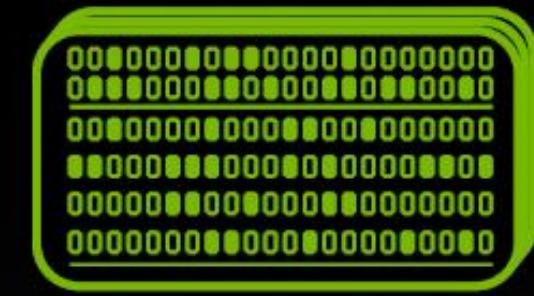
Programming the HPC Platform



NVC++



NVC



NVFORTRAN



NVCC

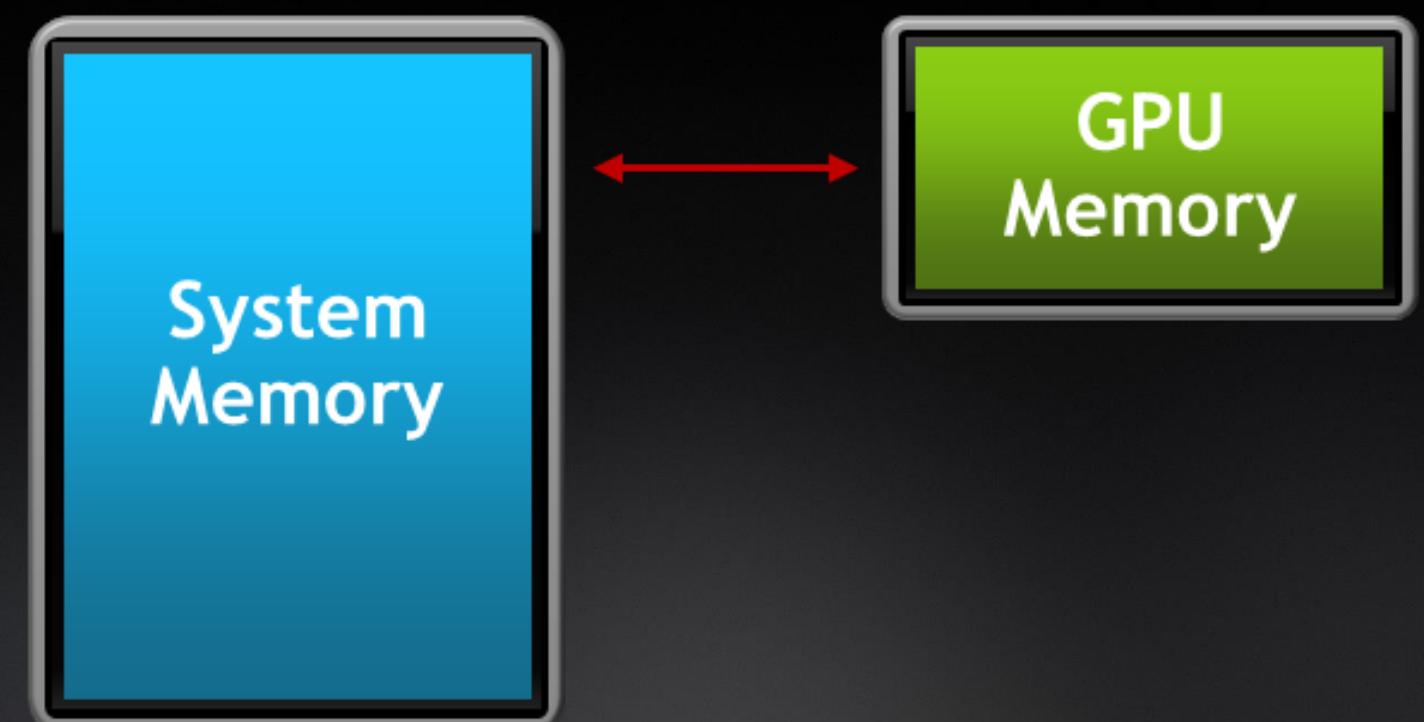
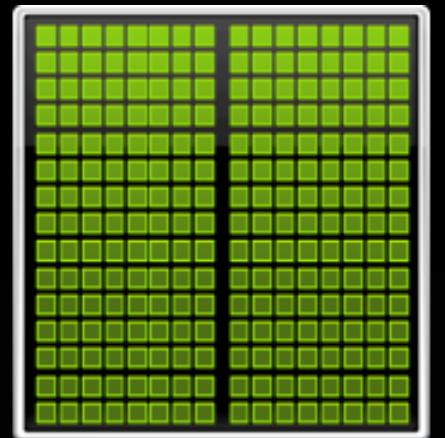
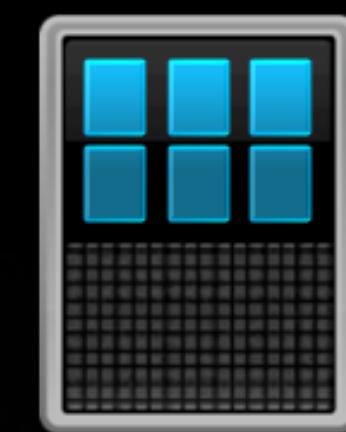
# NVIDIA HPC COMPILER

## Using OpenMP

- OpenMP
  - `-mp` → Enable OpenMP and target Multicore
  - `-mp=gpu` → Enable OpenMP and target GPU and Multicore
- GPU Options
  - `-gpu=ccXX` → Set GPU target
- Compiler Diagnostic
  - `-Minfo=mp` → Compiler diagnostic for OpenMP
- Environment variable for NOTIFY
  - `set NVCOMPILER_ACC_NOTIFY 1/2/3`

# KEY ASPECT OF GPU PROGRAMMING

## Separate CPU System and GPU Memories



# OpenMP

Fork-join model

Will your existing OpenMP  
code run on the GPU?

```
#pragma omp parallel for
for (int i = 0; i < N; ++i) {
    compute();
}
```

# GPU PORTING ADVICE FOR OPENMP PROGRAMMERS

Re-compile and run is not a performant solution

- **Enable more parallelism!**
  - Use collapse(N) clause on loops to increase parallelism
  - Parallelize inner loops
  - Re-order loops or transpose arrays to enable SIMD/SIMT accesses in outermost loops
- Restructure your code!
  - Replace critical sections with atomics
  - Remove all I/O statements
  - Remove memory allocation
- Use compiler feedback to identify and factor out unsupported or non-scalable OpenMP constructs and API calls



Parallelism, Parallelism, Parallelism ...



# NVIDIA OPENMP EXECUTION MODEL

# OPENMP MODEL

OpenMP Execution Mapping to NVIDIA GPUs and Multicore

omp target

→ Starts Offload

omp teams

→ [GPU] CUDA Thread Blocks in grid  
→ [CPU] num\_teams(1)

omp parallel

→ [GPU] CUDA threads within thread block  
→ [CPU] CPU threads

omp simd

→ [GPU] SIMDlen(1)  
→ [CPU] Hint for vector instructions

# Offload an OpenMP Program

**target** Starts the offload  
**teams** Creates teams

Set number of CUDA Thread Blocks  
No effect for CPU

Set number of CUDA Threads  
Set number of CPU threads

```
6 #pragma omp target teams num_teams(X) thread_limit(Y)
7 {
8     //compute-region
9 }
```

```
$ nvc test.c -mp=gpu -Minfo=mp
6, #omp target teams
6, Generating "nvkernel_main_F1L57_3" GPU kernel
      Generating Tesla and Multicore code
```



# PRESCRIPTIVE OPENMP EXECUTION MODEL

# OPENMP PRESCRIPTIVE MODEL

## Explicit Parallelism

Programmer explicitly specifies every action to be taken by the OpenMP compiler.

omp teams

omp parallel

omp distribute

omp for/do

other directives

→ distribute loop iterations across teams  
→ distribute loop iterations across threads

Need more parallelism to fully occupy GPU!

```
57 #pragma omp target teams distribute parallel for \
58                                     reduction(max:error)
59 for(int j = 1; j < n-1; j++) {
60     for(int i = 1; i < m-1; i++) {
61         Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
62                               + A[j-1][i] + A[j+1][i]);
63         error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
64     }
65 }
```

## Example

<b>target</b>	Starts the offload
<b>teams</b>	Creates teams
<b>distribute</b>	Workshare teams
<b>parallel</b>	Creates threads
<b>for</b>	Workshare threads

```
$ nvc test.c -mp=gpu -Minfo=mp
```

```
57, #omp target teams distribute parallel for
57, Generating "nvkernel_main_F1L57_3" GPU kernel
      Generating Tesla and Multicore code
      Generating reduction(max:error)
Loop parallelized across teams and threads(128), schedule(static)
```

# More Parallelism: How to Obtain?

1<sup>st</sup> Way

Use collapse(2) to cover loop

You cannot always use collapse  
If loops aren't strictly nested

```
57 #pragma omp target teams distribute parallel for \
58           reduction(max:error) collapse(2)
59 for( int j = 1; j < n-1; j++) {
60   for( int i = 1; i < m-1; i++ ) {
61     Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
62                           + A[j-1][i] + A[j+1][i]);
63     error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
64   }
65 }
```

```
$ nvc test.c -mp=gpu -Minfo=mp
```

```
57, #omp target teams distribute parallel for
      57, Generating "nvkernel_main_F1L57_3" GPU kernel
          Generating Tesla and Multicore code
          Generating reduction(max:error)
          Loop parallelized across teams and threads(128),
schedule(static)
```

# More Parallelism: How to Obtain?

2<sup>nd</sup> Way

Use **parallel for** on  
the inner loop

```
55 #pragma omp target teams distribute reduction(max:error)
56 for( int j = 1; j < n-1; j++ ) {
57 #pragma omp parallel for reduction(max:error)
58   for( int i = 1; i < m-1; i++ ) {
59     Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
60                           + A[j-1][i] + A[j+1][i]);
61     error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
62   }
63 }
```

```
$ nvc test.c -mp=gpu -Minfo=mp

55, #omp target teams distribute
      55, Generating Tesla and Multicore code
      Generating "nvkernel_main_F1L55_3" GPU kernel
56, Loop parallelized across teams, schedule(static)
57, Team private (j,error) located in CUDA shared memory
      #omp parallel
      57, Generating reduction(max:error)
      Loop parallelized across threads(128), schedule(static)
```

# More Parallelism: How to Obtain?

2<sup>nd</sup> Way

Use **parallel for** on  
the inner loop

For GPU target

GPU leverages **teams**  
parallelism. Great!

Enabled more parallelism  
here! Great for GPUs!

```
55 #pragma omp target teams distribute reduction(max:error)
56 for( int j = 1; j < n-1; j++ ) {
57 #pragma omp parallel for reduction(max:error)
58   for( int i = 1; i < m-1; i++ ) {
59     Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
60                           + A[j-1][i] + A[j+1][i]);
61     error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
62   }
63 }
```

```
$ nvc test.c -mp=gpu -Minfo=mp
```

```
55, #omp target teams distribute
      55, Generating Tesla and Multicore code
      Generating "nvkernel_main_F1L55_3" GPU kernel
      56, Loop parallelized across teams, schedule(static)
      57, Team private (j,error) located in CUDA shared memory
          #omp parallel
```

```
      57, Generating reduction(max:error)
          Loop parallelized across threads(128), schedule(static)
```

# More Parallelism: How to Obtain?

2<sup>nd</sup> Way

Use **parallel for** on  
the inner loop

For CPU target

You lost CPU parallelism  
on outer loop

parallel is in a loop,  
potential fork-join  
overhead

```
55 #pragma omp target teams distribute reduction(max:error)
56 for( int j = 1; j < n-1; j++ ) {
57 #pragma omp parallel for reduction(max:error)
58   for( int i = 1; i < m-1; i++ ) {
59     Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
60                           + A[j-1][i] + A[j+1][i]);
61     error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
62   }
63 }
```

```
$ nvc test.c -mp=gpu -Minfo=mp
55, #omp target teams distribute
      55, Generating Tesla and Multicore code
      Generating "nvkernel_main_F1L55_3" GPU kernel
      56, Loop parallelized across teams, schedule(static)
      57, Team private (j,error) located in CUDA shared memory
          #omp parallel
          57, Generating reduction(max:error)
          Loop parallelized across threads(128), schedule(static)
```

# More Parallelism: How to Obtain?

## 3<sup>rd</sup> Way

Use macros

CPU needs coarse-grained parallelism

GPU needs fine-grained parallelism

```
#ifdef TARGET_GPU
    #pragma omp target teams distribute reduction(max:error)
#else
    #pragma omp parallel for reduction(max:error)
#endif
for( int j = 1; j < n-1; j++ ) {
#ifdef TARGET_GPU
    #pragma omp parallel for reduction(max:error)
#endif
    for( int i = 1; i < m-1; i++ ) {
        Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
                                + A[j-1][i] + A[j+1][i]);
        error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
    }
}
```

Tuned very well for CPU and GPU targets  
Now you need two executables

```
$ nvc test.c -mp=multicore -Minfo=mp -o program_cpu.exe
$ nvc test.c -mp=gpu -Minfo=mp -DTARGET_GPU -o program_gpu.exe
```



# DESCRIPTIVE OPENMP EXECUTION MODEL

# OPENMP DESCRIPTIVE MODEL

Leverage NVIDIA Compiler Analysis!

Programmer specifies the loop regions to be parallelized to the compiler **not every action**

omp loop

- Parallelizes across teams and threads
- Creates additional parallelism

# START OFFLOADING ‘OMP LOOP’

Three Ways

## 1. `omp target teams loop`

- Recommended way
- You can use `num_teams` and `thread_limit` clauses

## 2. `omp target loop`

- Fully automatic
- You cannot use `num_teams` / `thread_limit`

## 3. `omp target parallel loop`

- Uses only threads, and doesn’t use teams
- Might be useful for light kernels

# 4<sup>th</sup> way and the Target Portable Solution for More Parallelism

Use `loop` wherever you want parallelism, the compiler assigns the parallelism binding

For GPU target  
Parallelism is on outer and inner loop! Great!

For CPU target  
Parallelism is on outer loop! Great!

```
57 #pragma omp target teams loop reduction(max:error)
58 for( int j = 1; j < n-1; j++) {
59   #pragma omp loop reduction(max:error)
60     for( int i = 1; i < m-1; i++ ) {
61       Anew[j][i] = 0.25f * ( A[j][i+1] + A[j][i-1]
62                             + A[j-1][i] + A[j+1][i]);
63     error = fmaxf( error, fabsf(Anew[j][i]-A[j][i]));
64   }
65 }
```

```
$ nvc test.c -mp=gpu -Minfo=mp -o program.exe
```

```
57, #omp target teams loop
      57, Generating "nvkernel_main_F1L57_3" GPU kernel
57, #omp target teams
      57, Generating Tesla code
      57, Loop parallelized across teams /* blockIdx.x */
      59, Loop parallelized across threads(128) /* threadIdx.x */
57, Generating Multicore code
      57, Loop parallelized across threads
```

# TUNING DESCRIPTIVE MODEL

Use **bind** clause to help the compiler

- **loop bind(teams)** → [GPU] CUDA Thread Blocks and Threads  
→ [CPU] Threads
- **loop bind(parallel)** → [GPU] CUDA Threads  
→ [CPU] Threads
- **loop bind(thread)** → [GPU] Single thread  
→ [CPU] Single thread  
(hint for vector instructions)

# Tuning omp loop

For GPU target:  
The compiler identified  
beneficial thread  
parallelism

```
42 !$omp target teams loop
43 do i = 1, SZ1
44   do j = 1, SZ2
45     do k = 1, SZ3
46       do l = 1, SZ4
47         do m = 1, SZ5
48           ! expensive computation codes !
49         enddo
50       enddo
51     enddo
52   enddo
53 enddo
```

Single source,  
single binary  
targets CPU and GPU  
without  
ifdef  
or metadirective

```
$ nvfortran test.f90 -mp=gpu -Minfo=mp

42, !$omp target teams loop
42, Generating "nvkernel_MAIN_F1L42_1" GPU kernel
42, Generating Tesla code
43, Loop parallelized across teams ! blockIdx%x
44, Loop run sequentially
45, Loop run sequentially
46, Loop run sequentially
47, Loop parallelized across threads(128) ! threadIdx%x
42, Generating Multicore code
43, Loop parallelized across threads
```

Tuning  
omp loop  
collapse(5)

```
42 !$omp target teams loop collapse(5)
43 do i = 1, SZ1
44   do j = 1, SZ2
45     do k = 1, SZ3
46       do l = 1, SZ4
47         do m = 1, SZ5
48           ! expensive computation codes !
49         enddo
50       enddo
51     enddo
52   enddo
53 enddo
```

All loops are collapsed

```
$ nvfortran test.f90 -mp=gpu -Minfo=mp

42, !$omp target teams loop
42, Generating "nvkernel_MAIN_F1L42_1" GPU kernel
        Generating Tesla code
43, Loop parallelized across teams, threads(128) collapse(5) !
blockidx%x threadidx%x
44,   ! blockidx%x threadidx%x collapsed
45,   ! blockidx%x threadidx%x collapsed
46,   ! blockidx%x threadidx%x collapsed
47,   ! blockidx%x threadidx%x collapsed
42, Generating Multicore code
43, Loop parallelized across threads
```

# Tuning omp loop

## Inner omp loop

First 3 loops are collapsed and distributed across teams

Inner 2 loops are collapsed and distributed across threads

```
42 !$omp target teams loop collapse(3)
43 do i = 1, SZ1
44   do j = 1, SZ2
45     do k= 1, SZ3
46       !$omp loop collapse(2)
47         do l = 1, SZ4
48           do m = 1, SZ5
49             ! expensive computation codes !
50           enddo
51         enddo
52       enddo
53     enddo
54   enddo
```

```
$ nvfortran test.f90 -mp=gpu -Minfo=mp
```

```
42, !$omp target teams loop
42, Generating "nvkernel_MAIN_F1L42_1" GPU kernel
        Generating Tesla code
43, Loop parallelized across teams collapse(3) ! blockIdx%x
44, ! blockIdx%x collapsed
45, ! blockIdx%x collapsed
47, Loop parallelized across threads(128) collapse(2) ! threadIdx%x
48, ! threadIdx%x collapsed
42, Generating Multicore code
43, Loop parallelized across threads
```

# Tuning omp loop bind(thread)

```
42 !$omp target teams loop collapse(3)
43 do i = 1, SZ1
44   do j = 1, SZ2
45     do k= 1, SZ3
46       !$omp loop bind(thread) collapse(2)
47         do l = 1, SZ4
48           do m = 1, SZ5
49             ! expensive computation codes !
50           enddo
51         enddo
52       enddo
53     enddo
54   enddo
```

```
$ nvfortran test.f90 -mp=gpu -Minfo=mp
```

```
42, !$omp target teams loop
        42, Generating "nvkernel_MAIN_F1L42_1" GPU kernel
          Generating Tesla code
        43, Loop parallelized across teams, threads(128) collapse(3) !
blockidx%x threadidx%x
        44, ! blockidx%x threadidx%x collapsed
        45, ! blockidx%x threadidx%x collapsed
        47, Loop run sequentially
        48, collapsed
42, Generating Multicore code
43, Loop parallelized across threads
```

First 3 loops are collapsed and distributed across teams and threads

Inner 2 loops are run sequentially

# PEARLS AND PITFALLS PRESCRIPTIVE VS DESCRIPTIVE

	Prescriptive 'omp distribute/for/do/parallel'	Descriptive 'omp loop'
Parallelism assignment	Programmer	Compiler & Programmer
Number of directives	Many	Few
Inner parallelism	omp parallel	omp loop
Single source CPU-GPU target performance portable	Not always	Yes
Using OpenMP directives: <b>master, single, barrier, critical, etc.</b> and OpenMP APIs	Allowed	Not allowed
Performance	Inner 'parallel' construct might introduce overhead	Full SPMD



# CALLING A PROCEDURE

# Calling Procedure Example-1

Use ‘declare target’ if it is in a different file

test1.c

```
#pragma omp declare target(dotp)
T dotp(T *a, T *b, int N) {
    T r = 0;
    for (int i = 0; i < N; ++i)
        r += a[i] + b[i];
}
```

test2.c:

```
#pragma omp declare target(dotp)
T dotp(T *a, T *b, int N);

void mvmul(T **a, T *b, T *res, int N) {
    #pragma omp target teams distribute parallel for
    for (int i = 0; i < N; ++i)
        res[i] = dotp(a[i], b, N);
}
```

```
$ nvc test1.c -c -mp=gpu -Minfo=mp
dotp:
 3, Generating device routine
     Generating Tesla code
```

```
$ nvc test2.c -c -mp=gpu -Minfo=mp
mvmul:
 10, #omp target teams distribute parallel for
 10, Generating Tesla and Multicore code
     Generating "nvkernel_mvmul_F1L10_1" GPU kernel
 12, Loop parallelized across teams and threads, schedule(static)
```

## Calling Procedure Example-2

Compiler detects functions if they are in the same compilation unit

```
test.c
T dotp(T *a, T *b, int N){
    T r = 0;
    for (int i = 0; i < N; ++i)
        r += a[i] + b[i];
}

void mvmul(T **a, T *b, T *res, int N) {
    #pragma omp target teams distribute parallel for
    for (int i = 0; i < N; ++i)
        res[i] = dotp(a[i], b, N);
}
```

```
$ nvc test.c -mp=gpu -Minfo=mp

dotp:
  5, Generating implicit device routine
      Generating Tesla code
mvmul:
  10, #omp target teams distribute parallel for
      10, Generating Tesla and Multicore code
          Generating "nvkernel_mvmul_F1L10_1" GPU kernel
  12, Loop parallelized across teams and threads, schedule(static)
```

# More Parallelism in a function

Orphaned loop

```
file.c:  
T dotp(T *a, T *b, int N){  
    T r = 0;  
    #pragma omp loop bind(parallel) reduction(+:r)  
    for (int i = 0; i < N; ++i)  
        r += a[i] + b[i];  
}  
  
void mvmul(T **a, T *b, T *res, int N) {  
    #pragma omp target teams loop  
    for (int i = 0; i < N; ++i)  
        res[i] = dotp(a[i], b, N);  
}
```

# LIMITATIONS OF ‘OMP LOOP’ IN NVIDIA HPC 21.3 COMPILER

## 1. Orphaned loop

- Allowed if callee-caller are in the same compilation unit
- Not supported if callee-caller are in different compilation units

## 2. Using parallel inside loop is not supported

- We are looking for a motivating example



# OPENMP DATA DIRECTIVES

map-types  
6 map-types

### **map(*to*: *list*)**

Allocates memory on GPU and copies data from host to GPU when entering region.

### **map(*from*: *list*)**

Allocates memory on GPU and copies data to host when leaving region.

### **map(*tofrom*: *list*)**

Allocates memory on GPU and copies data from host to GPU when entering region, and copies data to host when leaving region.

### **map(*alloc*: *list*)**

Allocates memory on GPU, no copies

### **map(*delete*: *list*)**

Removes the data on GPU memory, no copies

### **map(*release*: *list*)**

Decrement the present counter, no copies

target data

For structured data regions

```
real(8), allocatable :: x(:), y(:)
allocate(x(N), y(N))
```

```
!$omp target data map(x, y)
```

```
  !$omp target teams loop
    do i = 1, n
      call compute(x,y)
    enddo
```

```
  !$omp target teams loop
    do i = 1, n
      call compute2(x,y)
    enddo
```

```
!$omp end target data
```

# Using Managed Memory

NVIDIA compilers can automatically translate your dynamic data allocations to managed memory!

You don't need explicit `target data`

```
real(8), allocatable :: x(:), y(:)
allocate(x(N), y(N))
```

```
!$omp target data map(x, y)
```

```
!$omp target teams loop
do i = 1, n
    call compute(x,y)
enddo
```

```
!$omp target teams loop
do i = 1, n
    call compute2(x,y)
enddo
```

```
!$omp end target data
```

```
$ nvfortran test.c -mp=gpu -gpu=managed -Minfo=mp
```

enter data  
exit data

For unstructured data regions

```
class Matrix {  
public:  
    Matrix(int n) {  
        len = n;  
        v = new double[len];  
#pragma omp target enter data map(alloc:v[0:len])  
    }  
    ~Matrix() {  
#pragma omp target exit data map(delete:v[0:len])  
        delete[] v;  
    }  
  
private:  
    double* v;  
    int len;  
};
```



INTEROPERABILITY

# INTEROPERABILITY

- NVIDIA compilers support many programming models all of which are interoperable
  - OpenACC
  - CUDA C++/CUDA Fortran
  - Standard parallelism with Fortran DO CONCURRENT
  - Standard parallelism with C++ Parallel Algorithms
  - CUDA libraries: cuFFT, cuBLAS, and etc.
- Device data is shared between OpenACC and OpenMP

OpenACC  
OpenMP  
Fortran stdpar

Manage data with OpenACC  
Parallelize using all the models  
All runs on GPU

```
SUBROUTINE smooth(A,B,...)  
real, dimension(:, :) :: A, B
```

```
!$ACC DATA COPY(A,B)
```

```
DO iter = 1,niters
```

```
DO CONCURRENT(i=2:n-1, j=2:m-1)
```

```
    A(i,j) = w0 * B(i,j) + &
```

```
        w1 * (B(i-1,j) + B(i,j-1) + B(i+1,j) + B(i,j+1)) + &
```

```
        w2 * (B(i-1,j-1) + B(i-1,j+1) + B(i+1,j-1) + B(i+1,j+1))
```

```
END DO
```

```
!$omp target teams loop collapse(2)
```

```
DO i=2, n-1
```

```
    DO j=2, m-1
```

```
        B(i,j) = w0 * A(i,j) + &
```

```
            w1 * (A(i-1,j) + A(i,j-1) + A(i+1,j) + A(i,j+1)) + &
```

```
            w2 * (A(i-1,j-1) + A(i-1,j+1) + A(i+1,j-1) + A(i+1,j+1))
```

```
    END DO
```

```
END DO
```

```
END DO
```

```
!$ACC END DATA
```

```
END SUBROUTINE
```

Manage data with OpenACC

Parallelize with Fortran  
Standard Parallelism  
A, B ready in the GPU

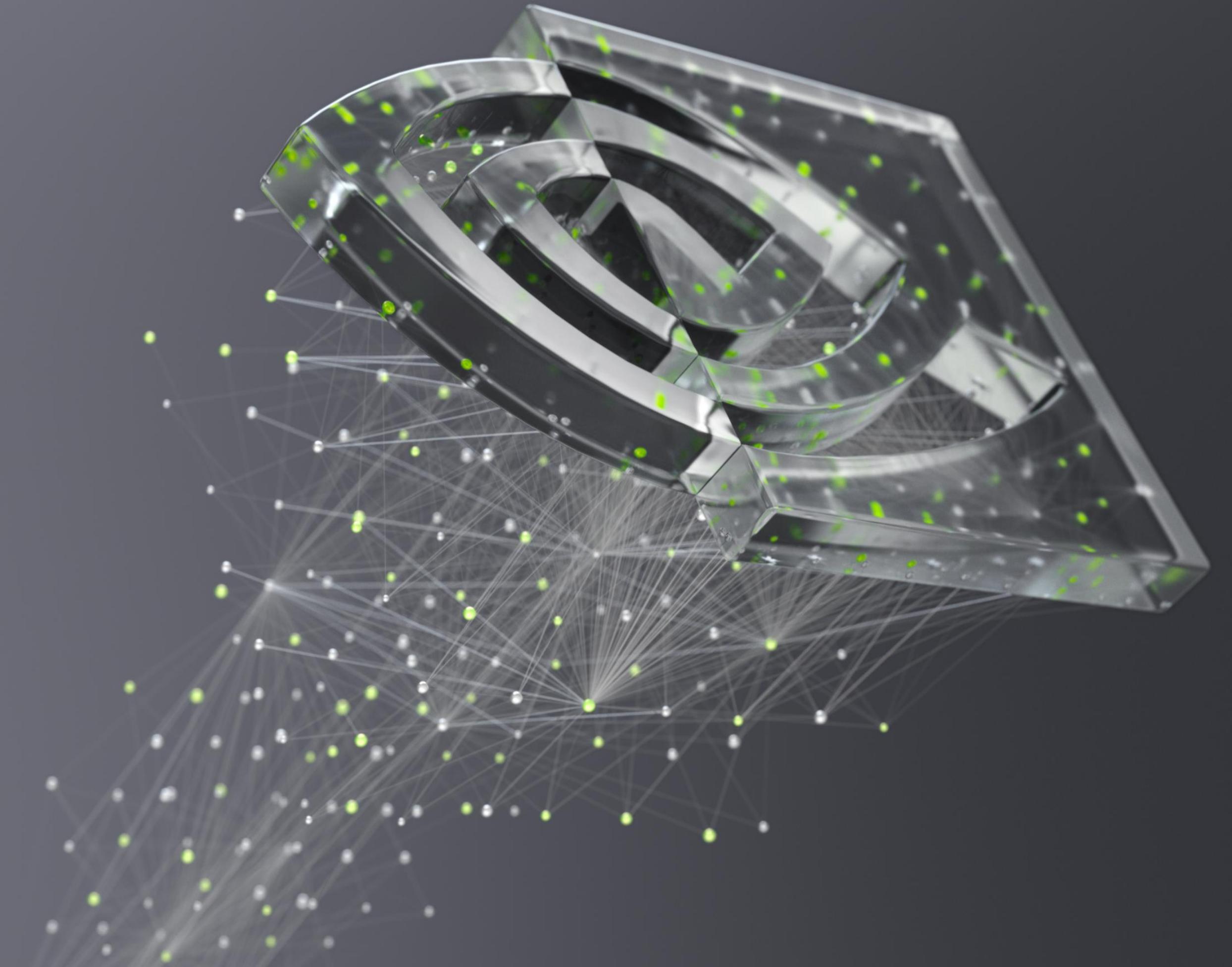
Parallelize with OpenMP  
A, B are ready in GPUs

```
$ nvfortran test.f90 -mp=gpu -acc=gpu -stdpar -Minfo
```

# THINKING OPENMP WITH NVIDIA COMPILERS

## Enable More Parallelism!

- First production release of OpenMP target offload in NVIDIA HPC SDK 21.5
- To take advantage, you need to enable more parallelism
  - Restructure your program: use collapse, use inner parallelism, do not use synchronization
- Two ways to enable more parallelism
  - Prescriptive model with ‘omp distribute/for/do/parallel’
    - Possible to parallelize everything
    - Requires a lot directives
    - Not always portable
  - Descriptive model with ‘omp loop’
    - Parallelizes fully parallel codes
    - Does not require a lot directives, leverages compiler analysis
    - Portable and Faster



NVIDIA®